What's in a Theorem Name?

What's in a Theorem Name? (Rough Diamond)

David Aspinall[1] and Cezary Kaliszyk[2]

[1] LFCS, School of Informatics, University of Edinburgh, UK.
[2] Institut für Informatik, University of Innsbruck, Austria.

Abstract. ITPs use names for proved theorems. Good names are either widely known or descriptive, corresponding to a theorem's statement. Good names should be consistent with conventions, and be easy to remember. But thinking of names like this for every intermediate result is a burden: some developers avoid this by using consecutive integers or random hashes instead. We ask: is it possible to relieve the naming burden and automatically suggest sensible theorem names? We present a method to do this. It works by learning associations between existing theorem names in a large library and the names of defined objects and term patterns occurring in their corresponding statements.

1 Introduction

How do we name theorems? Science has a tradition of *historical reference*, attaching names by attribution to their discoverer. HOL Light contains fine examples such as RIEMANN_MAPPING_THEOREM:

```
∀s. open s ∧ simply_connected s ⟺ s = {} ∨ s = (:real^2) ∨
    ∃f g. f holomorphic_on s ∧ g holomorphic_on ball(0, 1) ∧
          (∀z. z IN s ⟹ f z IN ball(Cx(&0), &1) ∧ g(f z) = z) ∧
          (∀z. z IN ball(Cx(&0),&1) ⟹ g z IN s ∧ f(g z) = z)
```

Mere lemmas are seldom honoured with proper names. Papers and textbooks use localised index numbers instead ("see Lemma 5.7 on p. 312") which is succinct but unhelpful. For practical proof engineering, working with large developments and libraries, we are quickly swamped with intermediate lemmas. Despite search facilities in some ITPs, users still need to name statements, leading to the proliferation of less profound *descriptive names* such as:

```
    ADD_ASSOC : ∀m n p. m+(n+p)=(m+n)+p
REAL_MIN_ASSOC : ∀x y z. min x (min y z)=min (min x y) z
   SUC_GT_ZERO : ∀x. Suc x > 0
```

Descriptive names are convenient and mnemonic, based on the statement of a theorem, and following conventions due to the author, library or system. But inventing names requires thought, and remembering them exactly later can be tricky. Unsurprisingly, people complain about the burden of naming, even inventing schemes that generate automatic names like (in Flyspeck [4]):

HOJODCM LEBHIRJ OBDATYB MEEIXJO KBWPBHQ RYIUUVK DIOWAAS

Such (fuzzy) hashes of theorem statements or sequential numbers give no clue of content, and we are back to old-fashioned indices. Some readers may be happy, accepting that names should have "denotation but not connotation" (as Kripke recalled the position of Mill [6]). But like Kripke, we see value in connotation: after all, a name is the handle for a potentially much longer statement.

So we wonder: *is it possible to relieve theorem naming burden by automatically naming theorems following established conventions?* Given that we have large corpora of carefully named theorems, it is natural to try a learning approach.

2 Parts of names and theorems

To start, we examine the form of human-generated descriptive theorem names. These are compound with separators: $l_0\_\ldots\_l_m$, where the l_i are commonly used stem words (labels). Examples like REAL_MIN_ASSOC show a connection between names of constants (MIN), their types (REAL), and the structure of the statement (ASSOC). Using previous work on features for learning-assisted automated reasoning [5], we extract three characterizations of theorem statements:

- **Symbols**: constant and type names (including function names);
- **Subterms**: parts of the statement term where no logical operators appear;
- **Patterns**: subterms, with abstraction over names of defined objects.

Patterns allow us to model certain theorem shapes, such as commutativity, associativity, or distributivity, without the actual constants these properties talk about [2]. For examples like SUC_GT_ZERO, we see that *where* the name parts occur is important. So we also collect:

- **Positions**: for each feature f, a position $p(f)$, normalised so that $0 \leq p(f) \leq 1$, given by the position of f in the print order of the statement.

The leftmost feature has position 0 and the rightmost 1; if only a single feature is found, it has position $\frac{1}{2}$. Names are treated correspondingly: for $l_0\_\ldots\_l_m$, the stems are assigned equidistant positions $p(l_0) = 0, \ldots, p(l_m) = 1$.

3 Learning associations between names and statements

We investigate two schemes for associating theorem statements with their names:

- **Consistent**: builds an association between symbols (e.g., constant and type names) and parts of theorem names;
- **Abstract**: uses patterns to abstract from concrete symbols, building a matching between positions in statements and name parts.

We hypothesise that the first scheme might be the more successful when used *within* a specific development, (hopefully) consistently re-using the same sub-components of relevant names, whereas the second may do better *across* different developments (perhaps ultimately, even across different provers).

To try these schemes out, we implement a k-Nearest Neighbours (kNN) multi-label classifier. A proposed label is computed together with a weighted average of positions. The algorithm first finds a fixed number k of training examples (named theorem statements), which are most similar to a set of features being considered. The stems and positions from the training examples are used to estimate the relevance of stems and proposed positions for the currently evaluated statement. The *nearness* of two statements s_1, s_2 is given by

$$n(s_1, s_2) = \sqrt{\sum\nolimits_{f \in \overline{f}(s_1) \cap \overline{f}(s_2)} w(f)^2}$$

where $w(f)$ is the IDF (inverse document frequency) weight of a feature f. To efficiently find nearest neighbours, we index the training examples by features, so we can ignore examples that have no features in common with the currently considered statement. Given the set of k training examples nearest to the current statement, we evaluate the relevance of a label as follows:

$$R(l) = \sum\nolimits_{s_1 \in N, l \in \overline{l}(s_1)} \frac{n(s_1, s_2)}{\left|\overline{l}(s_1)\right|}$$

We propose positions for a stem using the weighted average of the positions in the recommendations; weights are the corresponding nearness values.

Feature	Frequency	Position	IDF
$(V_0 + (V_1 + V_2) = (V_0 + V_1) + V_2)$	1	0.37	7.82
$((V_0 + V_1) + V_2)$	1	0.75	7.13
$(V_0 + V_1)$	1	0.84	3.95
$+$	4	0.72	2.62
num	3	0.21	1.15
$=$	1	0.43	0.23
$\forall$	3	0.15	0.03

Table 1. Selected features extracted from the statement of ADD_ASSOC.

As an example, we examine how the process works with the consistent naming scheme and the ADD_ASSOC statement shown previously. Our algorithm uses the statement in a fully-parenthesised form with types attached to binders:

$$\forall \mathsf{num}.(\forall \mathsf{num}.(\forall \mathsf{num}.((V_0 + (V_1 + V_2)) = ((V_0 + V_1) + V_2))))$$

From this statement, features are extracted, computing their frequency, average position and then IDF across the statement. A total of 46 features are extracted;

Theorem name	Statement	Nearness
MULT_ASSOC	(V0 * (V1 * V2)) = ((V0 * V1) * V2)	553
ADD_AC_1	((V0 + V1) + V2) = (V0 + (V1 + V2))	264
EXP_MULT	(EXP V0 (V1 * V2)) = (EXP (EXP V0 V1) V2)	247
HREAL_ADD_ASSOC	(V0 $+_{\mathbb{H}}$ (V1 $+_{\mathbb{H}}$ V2)) = ((V0 $+_{\mathbb{H}}$ V1) $+_{\mathbb{H}}$ V2)	246
HREAL_MUL_ASSOC	(V0 $*_{\mathbb{H}}$ (V1 $*_{\mathbb{H}}$ V2)) = ((V0 $*_{\mathbb{H}}$ V1) $*_{\mathbb{H}}$ V2)	246
REAL_ADD_ASSOC	(V0 $+_{\mathbb{R}}$ (V1 $+_{\mathbb{R}}$ V2)) = ((V0 $+_{\mathbb{R}}$ V1) $+_{\mathbb{R}}$ V2)	246
REAL_MUL_ASSOC	(V0 $*_{\mathbb{R}}$ (V1 $*_{\mathbb{R}}$ V2)) = ((V0 $*_{\mathbb{R}}$ V1) $*_{\mathbb{R}}$ V2)	246
REAL_MAX_ASSOC	($\mathrm{MAX}_{\mathbb{R}}$ V0 ($\mathrm{MAX}_{\mathbb{R}}$ V1 V2)) = ($\mathrm{MAX}_{\mathbb{R}}$ ($\mathrm{MAX}_{\mathbb{R}}$ V0 V1) V2)	246
INT_ADD_ASSOC	(V0 $+_{\mathbb{Z}}$ (V1 $+_{\mathbb{Z}}$ V2)) = ((V0 $+_{\mathbb{Z}}$ V1) $+_{\mathbb{Z}}$ V2)	246
REAL_MIN_ASSOC	($\mathrm{MIN}_{\mathbb{R}}$ V0 ($\mathrm{MIN}_{\mathbb{R}}$ V1 V2)) = ($\mathrm{MIN}_{\mathbb{R}}$ ($\mathrm{MIN}_{\mathbb{R}}$ V0 V1) V2)	246

Table 2. Nearest neighbours found for ADD_ASSOC.

Table 1 shows a selection ordered by rarity (IDF). The highest IDF value is for the feature most specific to the overall statement: it captures associativity of +.

Next, Table 2 shows the nearest neighbours for the features of ADD_ASSOC among the HOL Light named theorems, discounting ADD_ASSOC itself. Most of these are associativity statements. The stem AC_ is commonly used in HOL to denote associative-commutative properties.

Finally, the first predicted stems with their predicted positions are presented in Table 3. With ASSOC and ADD being the first two suggested stems, taking into account their positions, ADD_ASSOC is indeed the top prediction for the theorem name. The following predicitons are reasonable too:

AC_ADD_ASSOC, AC_NUM_ADD_ASSOC, AC_ADD, NUM_ADD_ASSOC.

Stem	Positions	Stem	Positions
ADD	[0.18; 0.14; 0.12; 0; 0.12]	MONO	[0.44; 0.49]
ASSOC	[1]	REFL	[1]
AC	[0; 1]	SELECT	[0]
NUM	[0.67; 0.45; 0; 0.70]	SPLITS	[0]
FIXED	[0]	RCANCEL	[1]
EQUAL	[0.25; 0.25]	IITN	[0]
ONE	[0]	GEN	[1]

Table 3. Stems and positions suggested by our algorithm, sorted by relevance measure.

Unsurprisingly, the consistent scheme performs less well in situations where new defined objects appear in a statement, it can only suggest stems it has seen before. The abstract scheme addresses this. For this, we first gather all symbol names in the training examples and order them by decreasing frequency. Next, for every training example we find the non-logical objects that appear, and replace their occurrences by object placeholders, with the constants numbered in the order of their global frequencies. For example, a name like DIV_EQ_0 becomes C_0_EQ_C_3, and the theorem statement is abstracted similarly.

For the statement of the theorem ADD_ASSOC the first three names predicted by the abstract naming scheme are: +_ASSOC, num_+, and num_+_ASSOC. The use of the stem ADD is only predicted with $k > 20$.

4 Preliminary evaluation

We perform a standard leave-one-out cross-validation to evaluate how good names predicted on a single dataset are. The predictor is trained on all the examples apart from the current one to evaluate, and is tested on the features of the current one.

For 2298 statements in the HOL Light core library, the results are presented in Table 4. We explored four different options for the algorithm:

- Upper: names are canonicalised to upper case.
- AbsN: we use the abstraction scheme described above for naming.
- AbsT: we use abstraction in statements before training.
- Stem: we use a stemming operation to break down names.

The first option is useful in the libraries of HOL Light and HOL4 where the capitalization is mostly uniform, but may not be desired in proof assistant libraries where this is not the case. For example min and Min are used with different semantics in Isabelle/HOL.

The last option allows us to model the naming convention in HOL Light where a statement is relative to a type, but the type name does not get repeated. For example, a theorem that relates the constants REAL_ABS and REAL_NEG can be given the name REAL_ABS_NEG rather than REAL_ABS_REAL_NEG.

Each row in Table 4 is a combination of options. Results are split in the columns: the number of statements for which the top prediction is the same as the human-given name (First Choice); the number where the human name is in place 2-10 (Later Choice); one of the ten names is correct modulo stem order (Same Stems); the number where the human-used stems are predicted but not combined correctly (All Stems); and the number for which at least part of the prediciton is correct (Stem Overlap). Altogether, the human-generated names are among the top ten predictions by some instance of the algorithm in over 50% of cases; 40% for the best performing version based on the abstract scheme. The abstract scheme beats the consistent mechanism even in the same library.

The final column shows the number of cases that fail completely. These include cases with familiar (historical) names such as:

- EXCLUDED_MIDDLE (proposed reasonable name: DISJ_THM)
- INFINITY_AX (proposed reasonable name: ONTO_ONE).

We would not expect to predict these names, unless they are already given in the training data. In other cases, failure might indicate inconsistency: the human names may not always be "correct". We uncovered some cases of inconsistency such as where multiplication was occasionally called MULT rather than MUL, for example.

Setup Options	First Choice	Later Choice	Same Stems	All Stems	Stem Overlap	Fail
-	118	533	180	911	516	40
Upper	134	583	182	901	474	24
AbsN	187	517	222	849	474	49
AbsN+Stem	218	530	208	849	460	33
AbsN+Upper	203	461	250	888	478	18
AbsN+AbsT	172	243	125	919	755	84
AbsN+AbsT+Upper	206	387	211	771	680	43
AbsN+AbsT+Stem	214	455	178	881	532	38
AbsN+Stem+Upper	238	491	299	835	418	17
AbsN+AbsT+Stem+Upper	273	501	291	757	459	17
Combined	336	728	271	632	321	10

Table 4. Leave-one-out cross-validation on the HOL Light core dataset.

5 Conclusions

The initial results are encouraging and suggest foundations for name-recommender systems that might be built into ITP interfaces. Even if the perfect name is not proposed, suggestions may spark an idea for the user. We plan to go further and look at case studies such as renaming Flyspeck, or using naming maps as a bridge between different ITPs. Moreover, more advanced machine learning schemes could be used to distinguish the use of the same symbol for different operators.

Related work. This appears to be the first attempt to mine named theorems and produce a recommender system for naming new theorems in a *meaningful* way. There have been a number of non-meaningful proposals, e.g., Flyspeck's random 8-character identifiers [4]; Mizar's naming scheme of theory names and numbers (examples like WAYBEL34:67 [3]); the use of MD5 recursive statement hashes [8].

Identifier naming has been studied in software engineering. Lawrie et al [7] investigated name consistency in large developments. Deissenboeck and Pizka [1] build a recommender similar to our consistent scheme (but using different methods). They note that programming style guides say identifiers should be "self-describing" to aid comprehension. Indeed, program obfuscators, intended to *hinder* comprehension, randomize identifiers, producing names like those in Flyspeck.

Acknowledgments. This work has been supported by UK EPSRC (EP/J001058/1) and the Austrian Science Fund FWF (P26201).

References

[1] F. Deissenboeck and M. Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.
[2] T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In *CICM 2014*, LNCS 8543, pages 267–281. Springer, 2014.
[3] A. Grabowski, A. Korniłowicz, and A. Naumowicz. Four decades of Mizar. *J. Autom. Reasoning*, 55(3):191–198, 2015.
[4] T. Hales et al. A formal proof of the Kepler conjecture. *CoRR*, 1501.02155, 2015.
[5] C. Kaliszyk, J. Urban, and J. Vyskočil. Efficient semantic features for automated reasoning over large theories. In *IJCAI 2015*, pages 3084–3090. AAAI Press, 2015.
[6] S. A. Kripke. Naming and necessity. In *Semantics of Natural Language*, pages 253–355. Springer Netherlands, Dordrecht, 1972.
[7] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? A study of identifiers. In *ICPC 2006*, pages 3–12. IEEE, 2006.
[8] J. Urban. Content-based encoding of mathematical and code libraries. In *MathWikis 2011*, volume 767 of *CEUR-WS*, pages 49–53, 2011.